CHAPTER 6

A Privacy Engineering Lifecycle Methodology

"They always say time changes, but you actually have to change them yourself."

—Andy Warhol

This chapter discusses a systems engineering methodology that can be adapted to privacy engineering. The methodology presented should be followed throughout development of a project for a privacy solution. It involves interactive models that provide pictorial documentation as well as business language use cases that together present requirements, analysis, design, and test cases in a readable form. The models work together to provide an understandable information and application architecture that satisfies business requirements, including, of course, privacy and security.

Executives may wish to glide through this chapter to get a feel how their teams work toward a project solution. Engineers, designers, and consultants will want to dig in deeper to perform their function more effectively.

The requirements use cases, the class model and supporting metadata, the user experience requirements, and any supporting requirements, as discussed in Chapter 5, are the basis for developing an architectural solution.

Enterprise Architecture

This section discusses an enterprise architecture approach that actuates these requirements into an architectural solution. The privacy engineering methodology is based on concepts derived from enterprise architecture.

An enterprise has been defined as an association consisting of a recognized set of interacting functions that are able to operate as an independent, stand-alone entity. There are enterprises within enterprises. For instance, a business unit within the overall corporate entity may be considered an enterprise as long as it could be operated independently.

Architecture provides the underlying framework, which defines and describes the platform required by the enterprise to attain its objectives and achieve its business vision. Architecture is an amalgam of engineering art and engineering science; there is no single enterprise architecture. Instead, the overall architecture can be considered to consist of four interrelated architectures or architectural views (Figure 6-1).

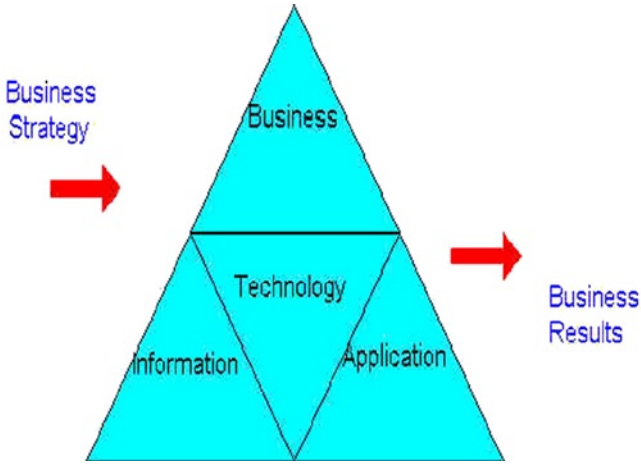

Figure 6-1. *Enterprise architecture views*

Architectural Views

The suggested architectural approach[1] envisions four architecture views: business, information, application, and technology architectures. These may contain levels of detail that are used to describe the elements of a privacy engineered architecture, but are not the processes themselves that will be built using these defined architectures.

In the privacy engineering methodology, the underlying architectural views have specific privacy opingcharacteristics:

- *Business architecture*: Models the business enterprise to show how business is to be done.[2] The use cases, activity diagrams, and supporting metadata documenting the business architecture privacy requirements are enterprise requirements that must be enforced.
- *Information architecture*: Enables the enterprise to develop a common, shared, distributed, accurate, and consistent data resource that is based on the various data models and supporting metadata. Some of the key factors in information architecture are privacy requirements. Data stewards[3] indicate that there are privacy requirements that need to be enforced based on their knowledge of the data for which they are responsible. This will take the form of a metadata indicator that shows that privacy rules need to be followed or that the data should be encrypted.

[1]The enterprise architecture section is based on two much-quoted papers: "Enterprise Architecture: What and Why" by Tom Finneran (www.tdan.com/i007ht03.htm) and "Enterprise Architecture: The What's And How's" by Tom Finneran (www.tdan.com/i018ht02.htm).
[2]Again, this is applicable to for-profit, nonprofit, and governmental enterprises.
[3]The privacy team will work with the data stewards to ensure that they are familiar with legal and enterprise privacy policies, procedures, and privacy rules.

- *Application architecture*: Links the information and business architectures to reflect applications and how they are used and distributed. The UML sequence diagram and the component diagram are application architecture documents. During the application architecture process, the architect determines whether to invoke privacy component rules or requirements from within the system or to use it as an app. The application architecture also reflects what privacy enabled technology (PET) components, if any, will be included in the design. PETs are discussed later in this chapter.
- *Technology architecture*: Links up with the application, business, and information architectures to provide interoperable technology platforms that meet the needs of the various user roles (actors) at identified work locations. In developing the technology architecture, decisions regarding which automated solutions can be employed and whether to build or buy them are made.

In addition to the four enterprise architecture views shown in Figure 6-1, there is another that can be considered.

- *User interface architecture*: Links up the information, business, application, and technology architectures with the user facing design and controls. The user interface architecture provides the user experience, as discussed in Chapter 5 and below. This type of architecture must provide a way to incorporate privacy requirements into the architecture and design of the user interface.

Solution Architecture

The solution architecture (Figure 6-2) is developed from a system engineering methodology that consists of joining a user interface architecture design, information architecture (reflecting data modeling and big data analysis), and an application architecture. Thus, the privacy engineer can draw from a known engineering design and build techniques to add fair processing requirements and standards in a manner that is readily understood. The first new step on the journey to privacy innovation begins on a well-trodden path.

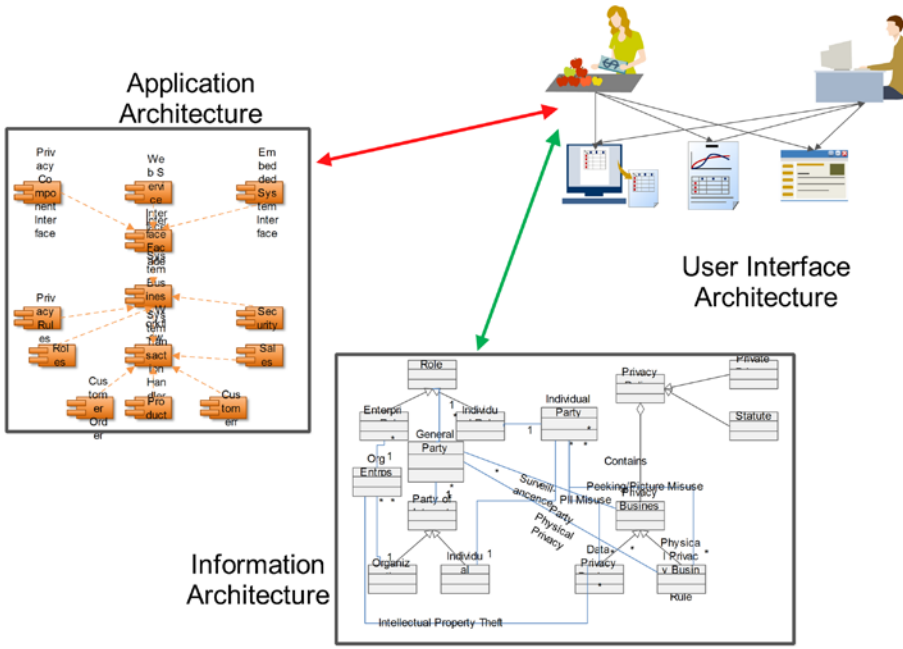

Figure 6-2. *Solution architecture*

Given the understanding of the business architecture, information architecture, and application architecture, the design team, including privacy engineering representatives, apply the appropriate technology architecture.

Develop Procedures, Processes, and Mechanisms

Privacy policy development is discussed in Chapter 4 and requirements development in Chapter 5. This chapter describes the methodology used to develop privacy procedures, processes, and mechanisms, focusing primarily on the latter (Figure 6-3). Note that mandated standards and recommended guidelines based on privacy policies heavily influence the end solution.

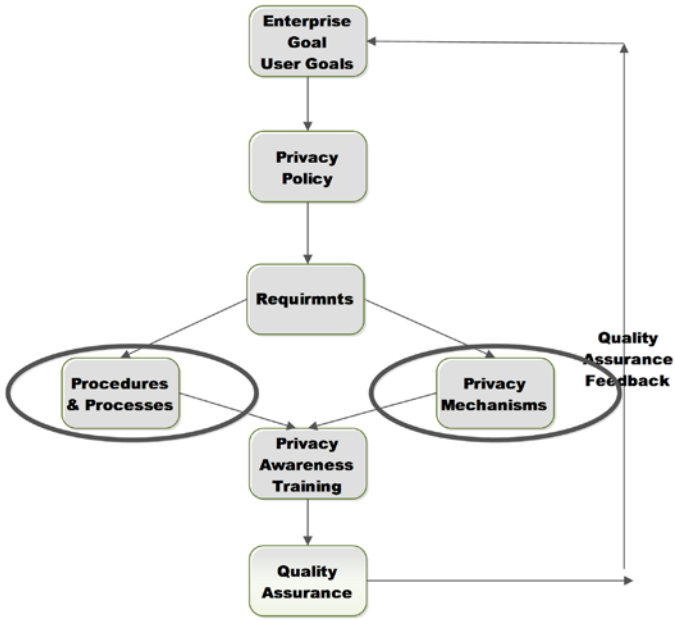

Figure 6-3. *Privacy engineering development process*

Methodology

System Engineering Lifecycle

Although the focus of this chapter is development of automated privacy mechanisms, the creation of processes and procedures will follow the same system engineering lifecycle (Figure 6-4). The system engineering lifecycle is a methodology that has commonly been used for at least 30 years. Some of the terminology has varied but the concepts remain the same. The familiarity of the design methodology makes it an excellent known best practice to leverage when adding in privacy specific requirements that may not have been raised at early phases of requirements gathering, planning, designing, and execution at the technical level.

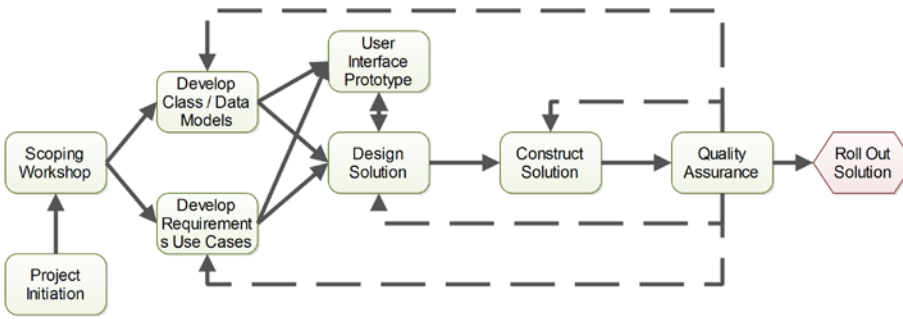

Figure 6-4. *System engineering lifecycle*

The system engineering lifecycle is composed of six stages in which the project team adapts the tools and methods to the environment in which they are working:

1. The project initiation and scoping workshop stages look at policies and best practices surrounding the enterprise and the expected project or projects being considered (see Chapter 4).
2. The development of requirement use cases and class or data model states defines the enterprise and seeks to understand the business requirements sought to be addressed (see Chapter 5).
3. The solution design stage includes prototyping the user interface for the project.
4. The implementation stage includes solution construction.
5. The quality assurance stage includes testing and user acceptance.
6. The final stage is solution rollout.

The lifecycle, at first glance, seems to be a "waterfalls approach," where one step is completed and then handed off to the next step until the project comes to completion, but the dashed feedback lines in Figure 6-4 show that the process is actually iterative. Incremental improvements will be made to project deliverables throughout the lifecycle. This methodology has been combined with Agile techniques on many successful projects. (See the sidebar "Privacy Engineering and Agile Development" to understand how this approach and Agile techniques can be integrated.)

It should also be noted that inclusion of privacy principles in the technology and governance frameworks early and continuously through the system engineering lifecycle returns added important utility. The governance framework or policies must be updated or managed as policies change. The resultant systems will be better understood and documented.

PRIVACY ENGINEERING AND AGILE DEVELOPMENT

Rich Schaefer - Director Technical Alliances, Good Technology

Various aspects of Agile development make it a very good fit for privacy engineering. A primary Agile tenet is to address customer needs by continually delivering working software that often must meet changing requirements. The customers for privacy engineering projects include internal and external stakeholders. Chapter 5 identified several actors present in use cases. The context diagrams in Chapters 7, 8, and 9 explicitly identify parties involved in the three scenarios. Notable are business stakeholders, especially the data stewards introduced in Chapter 3. As key members of the privacy team, they are both customers specifying privacy requirements and

participants in development. The original Agile principles[4] include a requirement that businesspeople and developers work together on a *daily* basis. Data stewards are the embodiment of the need for this type of collaboration. Their responsibilities include working directly with data analysts and database designers to develop data models.

Collaboration is inherent to privacy engineering, bringing together a wide variety of experts from different disciplines within business, privacy, information technology, and development. Agile project management approaches such as scrum can be used to bring effectiveness to such diverse teams. Agile scrums are mentioned as a best practice for coordinating privacy teams and their stakeholders to create and review metadata models in this chapter. Additionally, scrum meetings and sprints allow for timely adaptability to change. The need for flexibility to change is a recurring theme throughout this text. Privacy requirements can change due to factors external to the enterprise, including legal, consumer, and regulatory reasons. Within the enterprise, new business objectives, requirements, practices, and technology uses can have effects as well. Privacy engineering teams and their projects must be able to incorporate new requirements at nearly any point in their schedules. Agile processes enable the teams to prioritize changing requirements and even exploit such change for customer benefit.

The incremental delivery of working software via Agile sprints not only tries to guarantee that customers or their representatives receive what they desire, but also gives the opportunity for ensuring quality as the project progresses.[5] Regression testing at the end of each sprint may detect flaws that can be fixed within the following sprint(s). This practice avoids a shortcoming of traditional software approaches where quality assurance teams perform regression testing after development is completed and bugs are most costly to fix.

Given the general discussion above, one may ask how Agile engineering practices relate specifically to the formal techniques espoused in this text and depicted in the system engineering lifecycle (Figure 6-4). Use cases were introduced in Chapter 5 as the foundation for developing requirements for the system. They describe the needs of a user or actor and their answers to why, who, when, what, where, and how in describing the interaction within the system. Use cases can be seen as an agreement between customers and the development team.[6] Sufficient detail is provided for developers to understand what is required by the system and to embark on design.

User stories are a tool originating from the extreme programming (XP) Agile community for describing user needs and the planning of releases and iterations (their version sprints). Each consists of a few sentences, written in language a

[4]"Manifesto for Agile Software Development" at Agilemanifesto.org.
[5]The prototyping approach in this chapter is an example of incremental development.
[6]In fact, as mentioned in this chapter, first-cut use cases can be written by business users, with scrum interactions and a scrum review. This is not merely theory but fact at a major telecommunication company. These first-cut use cases could be considered user stories.

user could understand, expressing a single user's need representing an amount of work small enough to be reasonably well estimated. They serve as the basis for conversations with customers to flesh out more detail. Hence, use cases and user stories serve somewhat different purposes. However, accompanying user stories are acceptance criteria or tests that describe the conditions for their correct implementation. It has been noted that use cases and user stories plus their acceptance criteria are essentially equivalent. For much deeper comparison of use cases and user stories, see "Use cases vs. user stories in Agile development."[7]

UML models, also introduced in Chapter 5 for class and data models, give structure to the solution being developed throughout the system engineering lifecycle and provide an explicit communication tool among internal and external stakeholders. They have been applied for large enterprise teams and complex projects that have formal modeling methodology and documentation requirements. The Agile Manifesto values the interaction of individuals and working software over tools and comprehensive documentation. This apparently less formal approach has often led to the attitude that Agile methods are better suited to smaller projects and will not scale. However, significantly sized projects are referenced by Kent Beck (40 person-years)[8] and Scott Ambler (several hundred person-years).[9] Additionally, Agile modeling for scaling has been advocated by the latter, the developer of "Agile Model-Driven Development" based on Agile principles from XP.[10]

Agile proponents have had mixed reactions to the use of UML. Some say the practices within Agile development user stories and acceptance criteria supplant the need for UML. The most positive seems to be that UML should be used to work through specific issues where it is useful rather than in an end-to-end, comprehensive fashion. Martin Fowler's often-quoted article "Is Design Dead?"[11] discusses traditional planned design vs. evolutionary design employed by XP. He includes recommendations for the use of UML diagrams alongside "Class-Responsibility-Collaboration" cards typically used in XP. He emphasizes their use is for communication and can be used effectively for design exploration and documentation.[12]

---

[7]"Use Cases vs. User Stories in Agile Development" and the links within this article at www.boost.co.nz/blog/agile/use-cases-or-user-stories/.

[8]Kent Beck, "Test-Driven Development: By Example" (www.eecs.yorku.ca/course_archive/2003-04/W/3311/sectionM/case_studies/money/KentBeck_TDD_byexample.pdf).

[9]At a large-information-provider-over-200-person project, we used a combination of Agile and the UML-based approach discussed in this chapter. Chapter 9 is another example; well over 100 people were involved. Compare Chapter 8, where an intergenerational scrum was used along with UML modeling.

[10]"Agile Model Driven Development: The Key to Scaling Agile Software Development" at www.agilemodeling.com/essays/amdd.htm.

[11]"Is Design Dead?" at http://martinfowler.com/articles/designDead.html.

[12]The modeling, using UML, proposed throughout Part 2 of this book, is most effective where scrum-like modeling sessions and model review sessions, modelers, data stewards, privacy team, and other business stakeholders are held. In fact scrums have been used for good modeling before Agile and scrum terminology was being used.

Later in this chapter, creation of test cases from UML diagrams is described. A potential use of modeling, UML based or otherwise, has been proposed for the generation of test cases for tests-as-specification or test-driven development (TDD), a technique from XP.[13] TDD is iterative and proceeds by writing tests first and then developing code to pass the test. It produces simple code and is followed by continual refactoring or restructuring to avoid complexity and increase maintainability. TDD alone could be a good development process to employ in privacy engineering, because policy rules (e.g., in the privacy component) could be embedded in the tests driving the development and acceptance tests.

The software engineering lifecycle can incorporate either the formal use case or UML-based methodology in the text, employ Agile process management (e.g., scrum), use Agile engineering practices (e.g., from XP), or possibly a combination of these.

The Use of Models within the Methodology

The methodology utilizes a series of interrelated UML models, as shown in Figure 6-5.

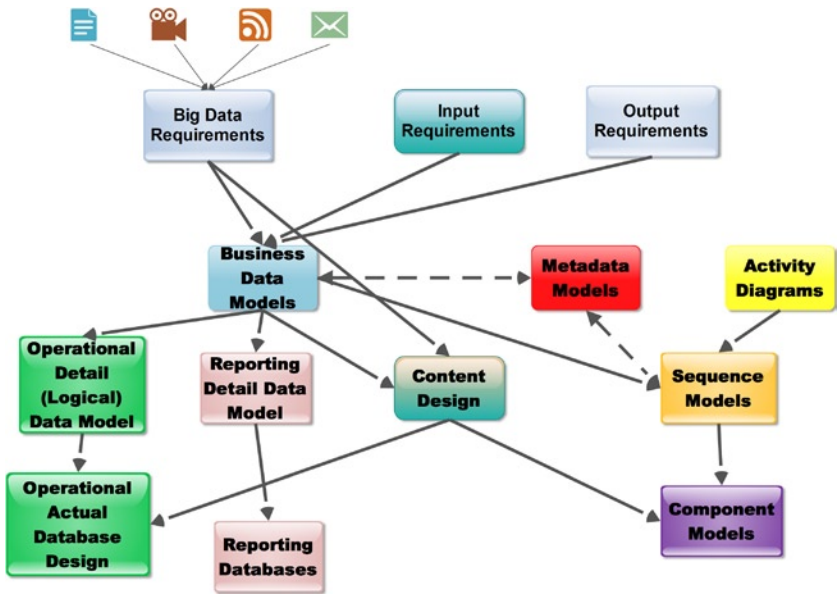

Figure 6-5. *Architectural model relationships*

[13]"Modeling in an Agile World" at www.nyu.edu/classes/jcf/CSCI-GA.2440-001/handouts/modellinginanagileworld.pdf.

Models and modeling best practices[14] focus first on the progression from an enterprise view of the business data model, through the more detailed logical data model, and finally to a database design based on these models. Likewise, from the business data model, a reporting model is derived for the reporting database.

Requirements models: Input and output data requirements gathered from a system interface, from a web site, or from a mobile source, together with the big data requirements must be modeled within the business data model, here using the UML class modeling diagram.[15] Figure 6-5 shows how the need for information from a document, from a video, from an audio file, from an e-mail, or from any other big data source comes together as big data requirements.

Business data model: The business data model is an integrated view of all of the data requirements within the enterprise. The business data model contains business-level (not necessarily normalized[16]) data classes. It may contain many-to-many data relationships and may not contain information about the optionality of data relationships. It should contain all super-type data classes but not necessarily all subtype data classes.[17] It will contain only those data attributes that are easy to find and define that are particularly interesting or important. It will refer to corporate data classes and relationships where possible and will raise data issues and ambiguities early.

Operational (logical) data model: The logical data model (see Figure 6-9 as an example) should contain all of the business data requirements within the problem domain under study (here, privacy information data processing). The conceptual data model subject areas, high-level data classes, and high-level relationships are used as the starting point for developing the logical data model. More detailed data classes are developed as well as data classes, which are the product of normalization. Subtype classes will also be derived from the high-level business data classes.

[14]See *Handbook of Relational Database Design* by B. Van Halle & C. Fleming (Addison Wesley, 1989), pp. 18–24; *Data Base Management* by F. McFadden and J. Hoffer (Benjamin Cummings Publishing, 1985), pp. 272–299; "The Bottom Line: Data-oriented Deliverables," by T. R. Finneran, in *Handbook of Data Management* (Auerbach Press, 1993), pp. 289–298.

[15]Other data modeling tools can be used. We recommend UML so that you can use one consistent toolset throughout the whole lifecycle.

[16]Normalization is a well-known data analysis process of organizing the data attributes to minimize redundancy and inconsistency. The business classes will not contain all of the data attributes and therefore normalization is not applicable. The logical data model will use normalization.

[17]Classes can be arranged in hierarchies so that concrete classes (subtypes such as persons or organizations) inherit attributes, relationships, and operations or methods from more abstract classes (super-types such as parties of interest).

The logical data model is different from the less-detailed business data model in that the former is normalized and does not contain many-to-many data relationships. Instead, it contains information about the optionality of data relationships and contains both super-type data classes and subtype data classes. It contains all data attributes relevant to the enterprise and refers to corporate data classes and relationships as much as possible.

As part of the data modeling process, the enterprise data model as well as legacy databases not represented in the enterprise data model will be examined to ensure that redundant data are not created and that the enterprise data models are complete.

Operational database: The detailed operational data model is used to develop the actual operational database. The reporting data models are used to develop the reporting databases, which could be the data warehouse, one or more data marts, or one or more big data analytic data structures. Big data requirements may also contribute to any required content handling or presentation.

Metadata models: All modeling metadata are based on a series of metadata models.[18] To ensure that models and modeling best support the corporate enterprise, best privacy engineering practices require that all models and modeling metadata be readily available to business users and to information technology personnel. All data administrators and database administrators should collaborate to ensure an enterprise view of all information required by the enterprise and to ensure that the best practices concerning shared data are followed.[19]

Best practices that support data sharing include data naming and data identification standards, the collection of integrity rules, the collection of security rules, and management of information in all of its forms. One way that works well in gaining collaboration among businesspeople, the privacy team, and the information technology development team is to hold Agile scrums. These scrums are often called first thing in the morning for the very detail-oriented people. Management scrums would be held weekly in some cases and biweekly in others.

[18]More than 20 metadata models comprise the database design of a typical metadata repository. Appendix A shows data attributes of some of these models.

[19]It must be noted that such collaboration does not require the mythical, monolithic data mapping and classification exercise of old where millions of dollars were expended, and consultants were sent swarming across the enterprise to arrive—perhaps—with a set of already outdated binders of data. Instead, data privacy principles define privacy information and a common understanding of how and where and by whom those data may be processed becomes a discovery methodology to evaluate existing data patterns.

Activity diagram, sequence model, and component model: The activity diagram showing the business process combines with the various data models to define a sequence within the system and then to the component design. The component design model and supporting metadata will contribute to the component design. Therefore, the various models and modeling efforts interact to provide a well-engineered, data-centric design.

Content Design: The user experience of the system and its user interface are based on the content design that takes inputs from the visualization aspects of the big data requirements and the business data model. The content design also impacts the actual operational database design and the component models by determining how users interact with them.

The steps of the methodology are described in detail in the following sections to illustrate how the system engineering lifecycle applied to privacy is effectively deployed.

INNOVATING WITH PRIVACY STANDARDS

By Dawn N. Jutla, PhD, Board Director, OASIS, and Professor, Sobey School of Business, Saint Mary's University, Halifax, Nova Scotia, Canada

Consumer and privacy legislators are working to understand new online business environments that exploit personal data outside of citizens' working knowledge and control. The Office of the Privacy Commissioner of Canada, the 27 different data protection agencies in the European Union, the US Federal Trade Commission, and senators in the US Congress now regularly question major innovators about their business practices concerning their handling of personal data. Associations such as the Electronic Frontiers Foundation and the Electronic Privacy Information Center also regularly highlight new online privacy violations. Media reports openly criticize marketers, raising awareness of personal data collection practices, as in the *Wall Street Journal*'s "What They Know Series": "Marketers are spying on Internet users—observing and remembering people's clicks, and building and selling detailed dossiers of their activities and interests."[20] VentureBeat, a technology news website, identifies a key privacy issue:

The fact of the matter is that most end users are ignorant of how much they expose about themselves when they authorize through Facebook or Twitter or any other sign-on process—and that this information would be shared to entities outside just the app developer.[21]

[20]"What They Know" (November 25, 2013). *Wall Street Journal*. Retrieved from http://blogs.wsj.com/wtk/.

[21]I. Mosquera (August 27, 2011). "Why Mobile Apps Need to Have Privacy Policies." *VentureBeat*. Retrieved from http://venturebeat.com/2011/08/27/why-mobile-apps-need-to-have-privacy-policies/.

To respond to this situation, can companies integrate privacy standards into Internet products and services to achieve an online environment that both protects privacy (as with user-permission-based models) and allows for commerce? OASIS (Organization for the Advancement of Structured Information Standards) is a leader in the Internet identity management and trust elevation standards space. Its OASIS Privacy Management Reference Model and Methodology[22] (PMRM) Technical Committee (TC) has created a committee specification draft as a standards track product.

The advantages of privacy standards are manifold. They include building a common and widespread understanding of privacy governance among adopting organizations at an international level and creating consistent compliance, auditing criteria, and user expectations across industries. Privacy standards can promote better system design, facilitate information interchange and interoperability, and foster innovation through multi-stakeholder collaboration. Some organizations may leverage the resulting privacy-enhanced products and services for market differentiation.

However, people don't usually think of standards as vehicles of innovation, even though numerous examples exist of new standards leading to new markets and technologies. Rather, standards are sometimes seen as the outcome of long political processes that are way too slow for young Internet innovators. These same innovators are busy with the newest commercial technologies, such as Big Data plays, the emerging Internet of Things, and attendant new business models focused on aggregating, interlinking, and monetizing personal data. Meanwhile, the tension between these new business models and the user's privacy rights is increasing with each passing day. Indeed, there is a growing sense among experts that many Internet companies, renowned for innovation and high levels of experimentation with new services, are not well versed in best practices for privacy governance. These relatively young companies, and many others, would benefit from more comprehensive privacy governance guidelines from the executive to the unit software testing levels. Here is where the patient process of standards can pay off to play a catalyst role in spurring responsible innovation and competitive advantage for many.

Upcoming privacy standards should foster another entire level of protection for consumer rights, as well. Privacy consultants praise the OASIS PMRM standards-track specification for codifying the processes for specifying privacy requirements. One excitedly said, ". . . it's better than the ad-hoc processes that are in my head. Now I have an explicit reference methodology that my clients are willing to invest in."

Certainly, the PMRM is valuable for its step-by-step guidelines and clear and concise identification of privacy domains, controls, and critical touch points—or leakage points—through which data flow. Privacy stewards and other stakeholders may use the PMRM to create a privacy management analysis for use cases. PMRM's methodology

[22]Privacy Management Reference Model and Methodology (PMRM), Ver. 1.0, March 2012, OASIS Committee Specification Draft. Retrieved from http://docs.oasis-open.org/pmrm/PMRM/v1.0/csd01/PMRM-v1.0-csd01.pdf.

extends to helping software engineers understand complex privacy requirements inherent in today's collaborative web-based systems. Indeed, stakeholders can use the methodology to perform thorough privacy management analyses in a wide variety of contexts, from executive management to unit-level software testing for privacy compliance.

Focusing entirely on the software engineering space is the work of an even newer standards committee, the OASIS Privacy-by-Design Documentation for Software Engineers Technical Committee[23] (PbD-SE TC), which I convened and co-chair with Dr. Ann Cavoukian, the founder of Privacy by Design, and Ontario's Information and Privacy Commissioner. The PbD-SE TC members are collaborating on a future standard that will help software engineers visualize privacy requirements and operationalize Privacy by Design principles. As a first step, the PbD-SE TC has accepted the PMRM specification to help organizations create use cases that embed privacy requirements as functional requirements. In addition, this TC is currently debating a new hybrid method of using software engineering modeling languages and spreadsheets to represent integrated privacy requirements in tabular and diagrammatic forms. Together, these approaches represent richer privacy models for our increasingly socially responsible software engineers.

As shown in this timely book, professional software engineers in industry use Unified Modeling Language (UML) diagram models for sharing vision, giving visual representations of (sub)-systems, influencing code generation, and documenting software requirements and design. The Object Management Group (OMG)'s UML is an International Standards Organization (ISO) software engineering industry modeling standard. Because of UML's ubiquity, OASIS PbD-SE leverages UML and may offer new extensions to it to support privacy.

Software engineers use UML to understand and collaborate on building software. UML abstracts away confusing details and allows software developers to more easily examine a system's behavior, data, and process models more quickly compared to textual documentation. However, while UML is a commonly used communications medium, it has different degrees of adoption and use. For some large systems, UML use may be quite formal, while for users of agile methodologies, software engineers may sketch out a quick UML-like diagram that allows them to share and easily refer to requirements and design. Today, requirements analysis takes up the largest proportion of time in agile software engineering efforts. Any aid in reducing the amount of time an engineer spends in understanding and embedding privacy requirements is a bonus for productivity. Hence, the work of the OASIS PbD-SE is positioned to provide such a productivity boost to the field.

[23]OASIS Privacy by Design Documentation Technical Committee (PbD-SE) Charter. Retrieved from www.oasis-open.org/committees/pbd-se/charter.php.

In summary, organizations participating in online privacy standardization efforts today provide valuable leadership in shaping tomorrow's privacy-preserving societies. Software engineers, from business analysts and software developers to unit testers, can use the current OASIS PMRM 2013 committee specification draft and the OASIS PbD-SE standards-track approaches to promote high quality privacy engineering and responsible governance.

Author's Note: The Privacy Engineering methodology described in this book is based on a system's engineering methodology used for over 30 years and therefore developed independently from PMRM and PbD, but when we reviewed these approaches, we found that privacy engineering is consistent with these approaches. We have been using UML from its early days. When Jonathan and Michelle presented their privacy assessment approach, we adapted it to UML using existing UML icons without extending UML. Dr. Jutla will be reviewing our proposed approaches as part of the OASIS PbD-SE TC analysis.

Stage 1: Project Initiation and Scoping Workshop

Project Initiation Defines Project Processes

During project initiation, the project team will develop project mechanisms for:

- Developing a first-cut project plan, including a statement of project objectives and scope. It should also include project tasks, resource roles, task start date and duration, and task dependencies.
- Defining the method for monitoring milestone deliverables.
- Reporting project status, including reporting period accomplishments, next period plans, problems or issues, and suggestions.
- Managing change or service requests.
- Release to management.

Change management is critical to the success of a project and must be fully formalized, approved, and promulgated via service requests. The change management process should be tracked and documented from the receipt of the first service request to the final implementation. Service requests should:

- Trigger all system development activities
- Be made for all scope changes that could affect a project's objectives
- Be made for all scope changes that will affect a deliverable's completion date

- Be analyzed in regard to impact of the project on the entire enterprise
- Have a measurable business benefit stated

Release management should provide a formal process for authorizing the movement from development and test into the production environment. Changes should be scheduled as releases, as much as possible, and the scope of next releases should be made available to all interested parties. The following steps should be performed:

- *Track problems and issues*: Issue number, related project, task problem or issue description, responsible team member, date reported, resolution, date closed, status, priority, reported by whom
- *Hold analysis, design, and development walkthroughs*: Management and technical team
- *Measure success and design metrics*: Process engineering metrics (mean time to failure, repair, and extend), deliverables delivered, resources to deliver
- Obtain user signoff on preagreed to measure of success

Requirements Definition Within the Scoping Workshop

To win a race, the s wiftness of a dart availeth not without a timely start.[24]

Fred Brook's classic article "The Mythical Man Month" begins with the following profound observation: "More software projects have gone awry for lack of calendar time than for all other causes combined. Therefore it is important to get a project off to a running start."[25]

John Zachman has stated that the beginning phase of any project is scoping objectives.[26] During the first week of any project, a scoping workshop is in order, during which a variety of business users, the privacy team, and information technology (IT) participants meet, preferably out of the office, to develop a project mission statement. A mixture of user executives, managers, the privacy team, and workers along with knowledgeable IT persons works best, but a less diverse group will be successful as long as the participants understand the business. The scoping workshop participants then develop a context diagram (see examples in Chapters 7, 8, and 9) that shows the suppliers and recipients of information from the engineered solution.

[24]Jean de La Fontaine, 1621-1695, *Fables* as quoted in L. D Eigen and J. P Siegel, *The Manager's Book of Quotations* (AMACOM, 1991).
[25]F. B. Brooks, Jr., *The Mythical Man-Month* (Addison-Wesley Publishing Company, 1975), p. 14.
[26]J. A. Zachman, "A Framework for Information Systems Architecture," in *Handbook of Data Management* (Boston: Auerbach Publications, Warren Gorham Lamont, 1993), pp. 3–22.

Next, the scoping session identifies major business classes, major business events, major business processes, major business rules, and major business objectives.

The participants then review and set study priorities on major business events, processes, or business classes. Typically, the events, processes, or business classes will be designated either as being a primary focus item, a secondary focus item, or out of scope. For primary and secondary focus items, stakeholders and subject matter experts are identified. The stakeholders and subject matter experts will be use case participants, those interviewed, or both.

Scoping Deliverables

The following deliverable may be developed[27] from the scoping workshop:

- List of business drivers
- Scoping mission statement
- Context diagram
- List of context actors
- List of actor locations
- List of triggering events
- List of information flows
- List of business classes
- List of business processes
- Potential privacy requirements
- Use case schedule using identified subject matter experts

Stage 2: Develop Use Cases and Class or Data Models

Chapter 5 discussed use cases in detail. This is the step in the methodology where use cases should be developed. In the following chapters, other use case examples are presented.

Develop Business Activity Diagrams

The business activity diagram in Figure 6-6 shows the events and processes and decision making between the various business processes involved in supporting vacation planning (Chapter 9 discusses the vacation planner example in scenario 3).

[27]These things come to the surface during the scoping workshop and may or may not be formally documented depending upon the time available.

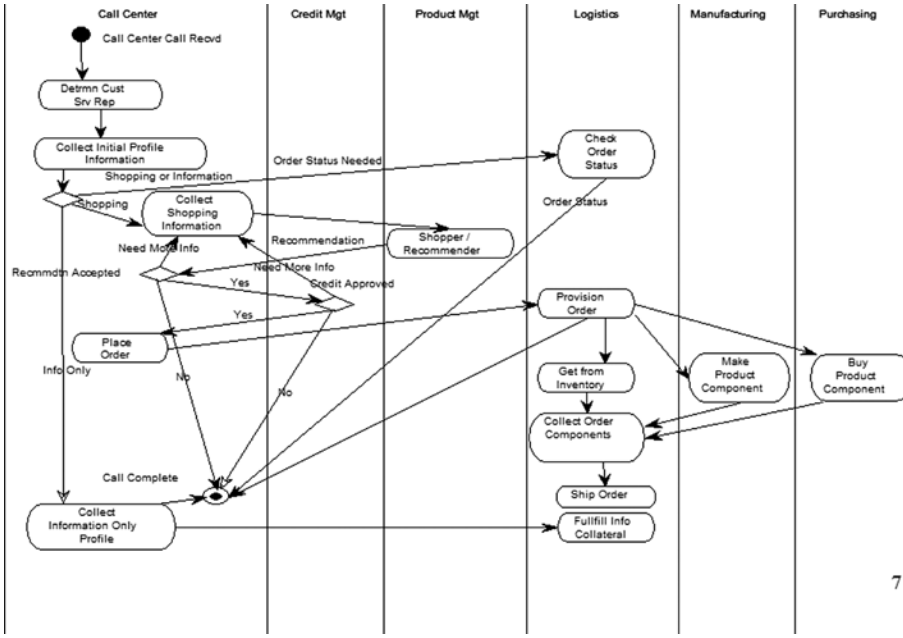

Figure 6-6. *Business activity diagram: vacation planning*

Using the Business Activity Diagram for Privacy Assessment

Some privacy professionals have proposed using the business activity diagram as part of the privacy requirements assessment. The privacy team works with business stakeholders, including data stewards, to identify key data attributes, especially identifiers, within the business processes and decisions, as represented in Figure 6-7. Privacy rules will be developed for these and other attributes as found and entered in the metadata.

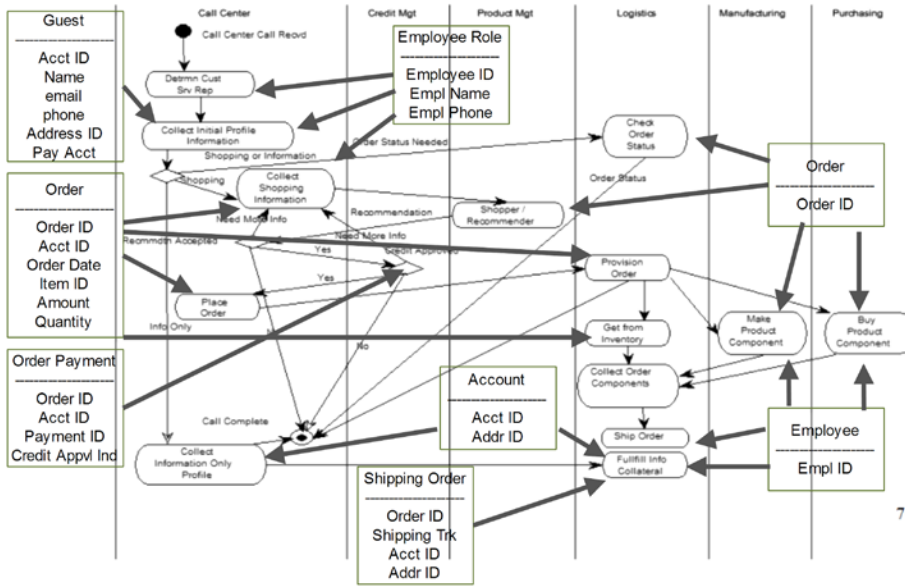

Figure 6-7. *Business activity diagram with key data attributes*

Defining Business and Privacy Data Classes

A *class* is a person, place, thing, concept, or event deemed to be of significance to an enterprise. Classes deal with attributes, behaviors, and message passing. A class has a name and a definition of its purpose and it has knowledge properties ("Data") and action properties ("Event Handlers and Processes").

Data classes can also be persons, places, things, concepts, or events of interest to the enterprise. Both class and data modeling approaches look at classes of things and how they are related to each other. During the business-level (conceptual) stage, methods (action properties) are not defined and the class model and data model may be congruent.

Where data required are contained within a document, such as a graphic, an audio input, web site content, something from e-mail, or from any other big data source, the data may be either processed as an object reflected in the data model as a data block or the data may be extracted within the program and stored as a data entity or data table. A data block shows the data attributes of the data class and would be processed using a NoSQL or a Hadoop system component. The scenario 3 vacation planner data model (in Chapter 9) shows an example of a big data data block within the data model.

The business data requirements are, perhaps, the most important requirements to be evaluated. If data are available in the database, a query can be developed to access it. If required data are not there, then significant customization is required. Business data modeling leads to a strong, well-designed, and flexible database.

Use cases identify classes and data attributes within the class. Class and data modeling support use case analysis. Class and class relationships are represented as UML class diagrams. Data and data relationships are represented as entity relationship diagrams.[28] Metadata document all aspects of class and data modeling. Data-oriented business and privacy rules are documented as metadata. (See Appendix A for examples of data-oriented metadata.)

Using the Unified Modeling Language Class Model as a Data Model

The class model, much like a data model, shows the information we manage and the relationships among the various classes. A data model reflects the data requirements and is the basis for the design of the database used to support the system meeting these requirements.[29] Each data item can have rules, identifiers, and universal truths that will become tables and columns within a database or otherwise processable data structure. These are the "things" we manage—policies, rules, people roles—when they turn into software or hardware. For consistency throughout the methodology, the UML class model is used for the data model.

One example class model is the party of interest model, which can be any individual or organization that is of interest to any enterprise. Figure 6-8 shows a more detailed piece of the class model that would be developed. The party of interest would have a uniqueness identification number, name, primary address, ZIP code, and primary telephone number. The relationship lines[30] indicate that persons and organizations are the most common types of party of interest and inherit the data attributes and the operational attributes (often referred to as methods). So person would have the attributes of party of interest as well as its own attributes. It would also have create, read, update, deactivate, and archive methods available.

[28]We use the UML class models for both class and data modeling. See the example below.
[29]We discuss a database here because it is in common use, but data models may be used in designing other data structures. Even in the case of unstructured data, data modeling helps organize the data elements extracted from the unstructured data into a "big data" data block. In the *Trillions* book (**Trillions: Thriving in the Emerging Information Ecology** by Peter Lucas, Joe Ballay, MickeyMcManus Wiley Press (2012)), the authors describe data storage containers that will implement a so-called internet of things. Understanding the various data entities, and the relationships of other data entities to it, is a condition precedent for the successful use of data.
[30]The arrow-like icon on the relationship lines indicates that there is an inheritance relationship between the super-type party of interest and the subtypes individual person and organization.

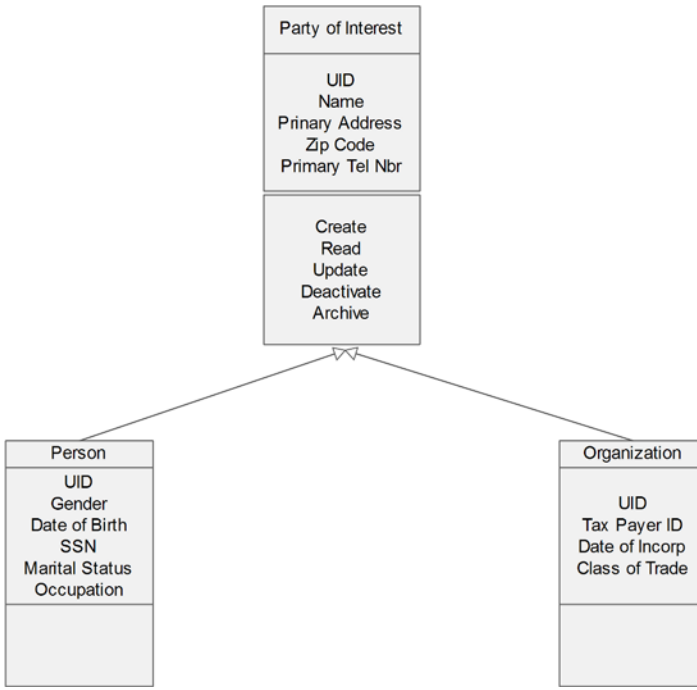

Figure 6-8. *Detail class model*

Example: Privacy Component Class Model

The party of interest is a class of the privacy component class model. Figure 6-9 shows *classes* that represent things managed by the enterprise and the data privacy requirements. Each class represents a person, place, thing, concept, or event deemed to be of significance to an enterprise within the data protection realm. Classes deal with attributes, behavior, and message passing. A class has a name and a definition of its purpose and other attributes that are characteristics of the class. This class model will be described in more depth in Chapter 7.

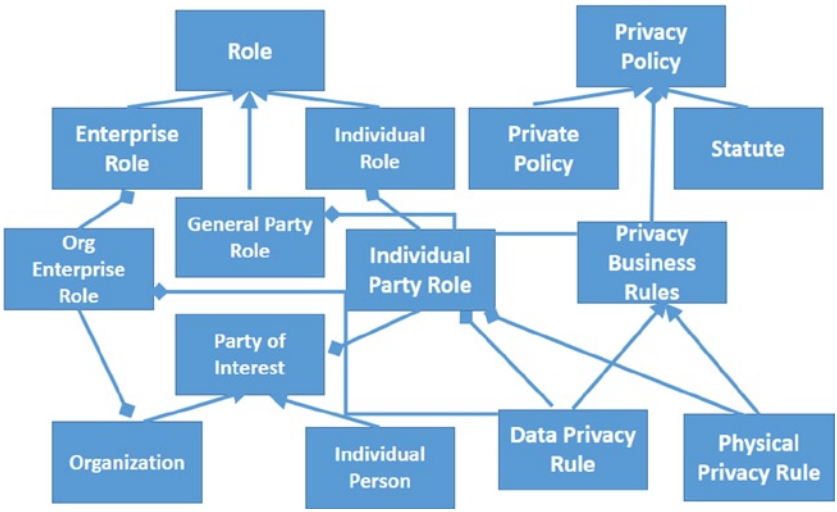

Figure 6-9. *Privacy component class model*

Data Modeling Steps

The following data modeling steps should be performed:

1. Identify major classes
2. Identify big data requirements (documents, videos, audios, web downloads, e-mails)
    a. Find where the useful data are located. In the case of big data, the same data may be scattered within and across different sources.
    b. Determine how to pull the data into a "single source of the truth" to consolidate, cleanse, and centralize the data.
3. Identify one or more data block(s) in which data attributes should be placed (e.g., a vacation plan data block in the scenario 3 vacation planner data model in Chapter 9).
4. Identify attributes of each class and big data data blocks
5. Determine relationships between classes and data blocks
6. Identify uniqueness identifiers (part of data modeling)
7. Validate classes through normalization and big data analysis

8. Attach business and privacy rules to classes, data blocks, data relationships, or data attributes
9. Integrate with existing class and data models
10. Analyze for stability and growth
11. Record in a metadata repository throughout process

Stage 3: Design an Engineered Solution

Once the analysis of business requirements has been completed, the project team works with the system developers to support the design of system solutions. The team will perform the activities in the following checklist, some of which are described in more detail in the following sections:

- Recommend redesign of business processes, where needed: Existing business processes that need to be revised and improved
- Define automation boundaries: Which business processes can be automated by technology and which processes are administrative
- Develop and utilize the system activity diagrams
- Expand system use cases and class models and supporting metadata
- Design the operational and reporting databases and big data analytics, from logical class or data model and expanded data models, including big data data blocks
- Perform dynamic modeling
- Define service components and supporting metadata, including big data handling components
- Perform system evaluation and prototyping (as needed)
- Define design units based on use cases
- Design presentation layer (user interface), including any content handling or presentation
- Perform development and proof of concept prototyping (if needed)
- Design batch program modules
- Finalize the solution (application and technology) architecture

User Interface Design

Basic User Interface Design Steps

There are several user interface (UI) design best practices that should be followed:

1. Understand your users' requirements:
    a. What are they trying to accomplish?
    b. How experienced are the users?
    c. What interfaces are they used to?
    d. What data attributes to be collected or reported upon require special privacy rules?
2. Use UI patterns that are as familiar as possible to the users.
3. Recognize a data hierarchy. For instance, an order with descriptive information about the order and one or more items should be shown as the order description with a list of clickable items that, when clicked on, will give a description of the selected item.
4. Interact with the user:
    a. Be as self-descriptive as possible
    b. Provide feedback
    c. Help users and forgive mistakes
    d. As the user becomes more used to the system, allow the user to select a more powerful, sophisticated interface
    e. Keep interactions conversational
    f. KISS (Keep It Simple, Stupid)

Mapping Business Class Objects to System and Technology Objects

The UI can be designed by mapping the business class objects to the system and technology objects.

There are various types of business class objects

- *Elemental business object*:[31] Class and related components and relationships
- *Complex business class object*:[32] User view and related components and relationships, including big data object.
- *Atomic business object*: Data attribute and related components and relationships

System objects are business objects viewed from a system's perspective. There are various types of system objects:

- *Elemental presentation objects*: Forms, lists, reports, or graphics of elemental business objects
- *Complex presentation objects*: Forms, lists, reports, or graphics of complex business objects, including privacy notices
- *Action selection mechanisms (controls)*: Icons, pop-up or pull-down menus, pop-up or pull-down lists, action buttons, radio groups
- *Specific functional object modules*: Ad hoc reports and queries, security, configuration management, privacy notice presentation mechanisms, and consent mechanisms (opt-in or opt-out).

User Interface Prototype

A crucial part of rapid application design and development is development prototyping, which is performed by the development team, consisting of IT personnel and business personnel. At the minimum there should be a team leader, prototype developers, and a modeler, along with representative business knowledge workers. In the case of a privacy-related project, the privacy team should be represented. It cannot be overstated how important the role of a great user interface designer who is skilled in aesthetic, functional, and technical aspects of user based interfaces can be. Because privacy engineering is relatively new and certainly rarely practiced, the more user centric and less opaque or "creepy" intrusive the interface, the more acceptable and the more data or person centric the system end product will be.

Larger functional areas will require more people. Starting with the demonstration (analysis) prototype, the online system is developed interactively with the business knowledge workers, along with further reports and functionality invoked by means of the system's presentation layer. Analysis and development prototyping are similar in

[31]Elemental objects or classes may be considered analogous to data classes. Elemental objects are analyzed utilizing an approach called "fact-based normalization."
[32]Complex objects are objects comprising or using information from more than one elemental object.

method. Development prototyping is more design-oriented and, thus, more detailed. The development prototyping deliverables are:

- A working prototype of the online application system
- A portion of the system design
- Detailed information required to transform the logical data model and the system use cases into the implementation system and the implementation database

Prototyping Caveats

Prototyping is inherent in the design approach described previously. However, no matter how good the development team's efforts are and no matter how good the prototype looks and acts, *the prototype is NOT the production system.*

- The team does not take time to tune the prototype for performance.
- Entity and referential integrity protection may not be completely developed.
- Although some of the security features may be developed in order to demonstrate how security might work, the security system, especially security administration, will not be completely developed.
- Although some of the help screens may be geared toward the business knowledge workers, the help system and screens will not be complete.
- Although the most important exception processing will be developed and demonstrated to the business knowledge workers, not all exception processing will be completed.
- Some of the system administration functionality, especially crucial reference tables, will be designed and geared toward the business knowledge workers, but not all system administration will be completed.
- Some stress testing experiments will be carried out in regard to the server and the network as a part of proof of concept prototyping. The remainder of stress testing will take place once production development is completed.

Component Design

What Is Component Architecture?

A component architecture[33] is a representation of the underlying set of interrelated components that define and describe the solution domain required by the business to attain its objectives and achieve its business vision.

COMPONENT ARCHITECTURE HISTORY

By Tom Finneran

In this book, we propose using a component architecture approach. There might be some concern that designers might try a noncomponent architecture approach. However, one might make the case that all programmers make use of components and component architecture.

From the beginning of computer programming, we programmer designers have grouped our code into modules or subroutines. We might have an input code module, a process module, and an output module. In the 1830s, Ada Byron Loveless, studying the Babbage Differencing Engine, developed an algorithm for calculating a sequence of Bernoulli numbers. The algorithm contained an input module, a processing of the numbers module, and a resulting list of numbers. Even back then, we can consider the modules a type of component, and Ada's approach was an early form of component architecture.

A component is a self-contained, reusable building block that can be used independently or assembled with other components to satisfy software requirements. A component handles a specific event, or related set of events, and provides a particular function or group of related functions through a well-defined and stable interface. All components consist of one or more component interfaces, component decision event handlers, and component behavior activators. The component interface may send or receive data from a file or may be a user interface. The decision event handler utilizes business rules to determine which component behavior should be activated.

It is important to understand that from the beginning of computer programming some form of component identification and architecting was done, although the terminology was developed later. Things like routines, subroutines, macros, and subsystems can be considered forms of a component.

[33]See "A Component-Based Knowledge Management System" by Thomas R. Finneran at www.tdan.com/i009hy04.htm.

INVENTION METHODOLOGY

By Tom Finneran

I was approached by a team of engineers who had an invention idea. It was a network interface card (NIC) based on a standard network protocol that would greatly increase the power of a local area network (LAN). We started with a scoping workshop, as discussed above. We then worked up a set of use cases and then developed a component architecture, based on the component architecture metadata model. This gave us an engineering spec from which we could designed both the card and supporting software, but also having engineering documentation, which made a favorable impression on the companies to which we were presenting our invention. The documentation was mapped right on to the patent application, including the patent claims, which are the basis for any patent.

So the methodology led to a hardware/software solution and a very straightforward patent approval process. See Patent #60/029,902.

Example: Privacy Component

We can understand how the privacy component, or any component, might work by interpreting the component metadata model (Figure 6-10). The privacy component may be embedded in a system or as a mobile app or web service or program subroutine. It may invoke a more broad-based system in the Cloud.

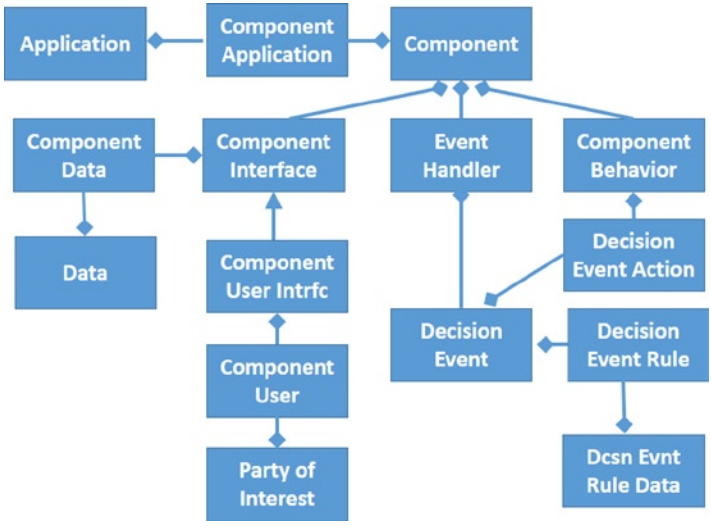

Figure 6-10. *Component metadata model*

The component interface may utilize a database based on a data model and, in most cases, it may utilize the database for the system it is embedded within (e.g., the simplified customer order data model in scenario 3 in Chapter 9). The component interface may also have a user interface for interacting with the actors shown in a context diagram.

The component event handler will process the events listed as the triggering events in the use case requirements (see Chapter 5). Each event implies one or more decision to be made. For instance, the UI will ask the user if he or she wishes to see the Privacy Notice.[34] When the user answers, an affirmative answer invokes one set of privacy rules and a negative answer invokes another set of privacy rules. Thus, each event triggers one or more decisions, and each decision requires a set of privacy rules as the criteria for making the decision. Each decision will then invoke a process or behavior that may trigger another event, and its decision sets or may invoke another behavior, all in accordance with the business rules. Each of the rules may require access to a database related to the component.

Privacy Rules

A *privacy rule* is a type of business rule. A *business rule* is a written statement in natural language that functions as a communication tool to express a rule, decision criteria, or a policy common practice as a statement that relates to a decision involving business information or business processes. A business rule is represented as an IF . . . THEN . . . ELSE pattern.

For example, IF Privacy Notice is clicked THEN invoke Privacy Notice routine ELSE check user role routine. These privacy rules will be derived from the privacy policies and the privacy procedures, standards, and guidelines as discussed herein.

Develop a System Activity Diagram

A system activity diagram shows how the various actors impacted by the system interact with the system processes, which are program modules with components and subcomponents. In the privacy engineering methodology, we have added a new feature, using a UML Note icon, to show which module satisfies the various FIPPS or GAPP principles. Chapter 7 presents an example of a system activity diagram with the privacy engineering enhancement.

Dynamic Modeling

For event-triggered activity identified within each system use case, a UML sequence diagram is used to model implementation details of the various activities or transactions of the system. The sequence diagram represents an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result.

[34]FIPPS/GAPP requires that a Privacy Notice that defines the enterprise's privacy policies be made readily available to a system user.

A sequence diagram shows objects involved in the activity by vertical lines, which are called object lifelines or swim lanes. Horizontal vectors between the object lifelines represent the messages passed between the objects. The messages are drawn chronologically from the top of the diagram to the bottom; the horizontal placing or spacing of objects is arbitrary.

A message from one object to another can be defined by the method called, or invoked, by the sending object on the receiving object. The method called must belong to the definition of the class instantiated by the receiving object.

During dynamic modeling, methods are included in classes in the class model.

Figure 6-11 presents a simplified UML sequence diagram showing the entry of a scenario 3 order that shows use of the privacy component.

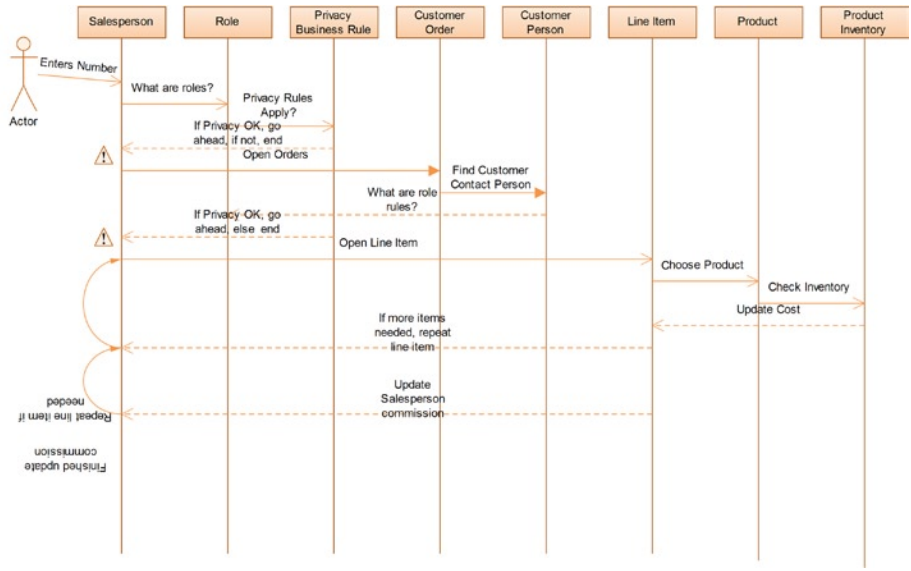

Figure 6-11. *Customer Order Sequence Diagram*

Define Service Components and Supporting Metadata

A service component is a self-contained, reusable building block component. It can be used independently or assembled with other components to satisfy an enterprise's requirement(s). A service component may implement one or more class objects and handles a specific event or a related set of events. It provides a particular function or group of related functions. A service component has a well-defined and stable interface(s).

UML defines a component as a software module (source code, binary code, executable, DLL, etc.) with a well-defined interface. The interface of a component is represented by one or several interface attributes that the component provides.

Components are used to show compiler and runtime dependencies as well as interface and calling dependencies among software modules. Components also show which component implements which specific class(es). Both business service classes (e.g., the customer class and the customer credit class) and controller classes (e.g., system business workflow class) may be considered part of a component within the component model (Figure 6-12). A UML component might not be a service component, in that the UML component may not meet the more rigorous service component definition stated previously.

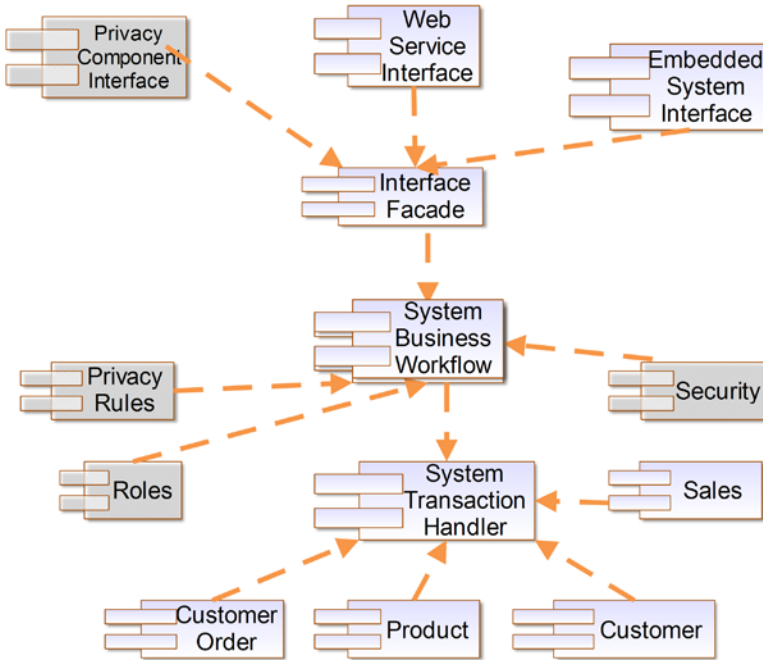

Figure 6-12. *Sample component diagram*

The privacy component can be seen as a component containing subcomponents. For instance, although Figure 6-12 would reflect a simplified component design for example scenario 3 (the vacation planner in Chapter 9), that scenario contains the embedded privacy component (for example scenario 1). The privacy component interface, the privacy rules, the roles, and the security components on the diagram are actually subcomponents of the privacy component.

Privacy Enabling Technologies

There is no uniform definition of PET; but it typically refers to the use of technology to help achieve compliance with data protection legislation or privacy policies. Many of the technologies referred to as PETs can protect corporate confidential information and protect revenues by securing the integrity of data. There are many PETs, and their benefits are both technology-specific and application-specific. The privacy component is itself a PET (Figure 6-13). The following concepts have been identified as PETs:[35]

- *Encryption*: Encryption may be implemented as a piece of code included in an information system or as a component invoked by the privacy component or by an embedded system.
- *Digital rights management*: Digital rights management (DRM) is a systematic approach to protect an enterprise's content and intellectual property. DRM technology focuses on making it impossible to steal content in the first place, a more efficient approach to the problem than the hit-and-miss strategies aimed at apprehending online poachers after the fact. Like encryption, DRM may be implemented as a piece of code included in an information system or as a component invoked by the privacy component or by an embedded system.
- *Privacy rules within application programs*: As discussed previously, privacy rules should be developed in conjunction with data stewards. System developers will implement those rules within the programs they develop. With the privacy component, privacy rules can be maintained easily and, if invoked by the various application programs, roles will be consistent throughout the enterprise. If the privacy rules change, those changes may be made within the privacy component and reflected within all of the various application programs. The changes are made in one place as opposed to individual changes made to all of the application systems.
- *Identity management*: Enterprises may develop identifiers for the various individuals impacted by their systems. Thus, they can develop a set of security components for authenticating their system users. There will also be authorization components that answer the question whether the user has the right to perform the action he or she is attempting. This may be based on security rules, privacy rules, or both. Authentication should be consistent throughout the enterprise. Therefore, including the authorization and authentication component as a part of the privacy component is often a prudent design decision.

[35]See "An Introduction to Privacy Enabling Technology" by Steve Kenny–Privacy Advisors, at https://www.privacyassociation.org/publications/2008_05_introduction_to_privacy_enhancing_technologies.

- *Engineering and architecture*: A well-architected system that utilizes the privacy engineering approach can be considered a form of PET.
- *Privacy information services*: As discussed previously, privacy information services can be considered a PET that can be plugged in wherever personal information functionality is needed.

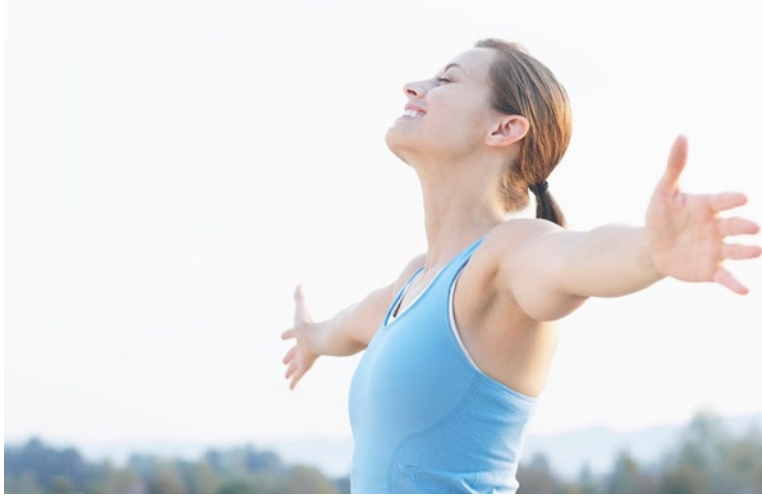

Figure 6-13. *PETs does not equal privacy*

Some feel that just by using PETs, they are protecting privacy. Although this can be partially true, it is not completely true. There is more to it than that. As discussed, a privacy solution may include PETs, for example, encryption, as one or more component within a component's architecture design. Even if the design is full of PETs, privacy will not be fully protected without well-written policies, standards, procedures, guidelines, and a notice presented in a readable form, among other things. PETs are enablers, but they are not substitutes for privacy engineering. PETs can be just one of many design components but alone are not a privacy solution.

BIG DATA: WHAT'S NEW? WHAT'S NOT? WHAT IT MEANS TO YOU: 10 THINGS YOU NEED TO KNOW

Leslie K. Lambert

Chief Security & Strategy Officer for GuruCul Solutions

1. **What is Big Data, *Really*?** Big Data is a term recently coined within the information technology field to describe tremendously large amounts of unstructured, or partially structured data that has been collected. Data is typically considered to rise to the level of "Big Data" when the amount of data that's available would take too much time and would cost too much money to load and process in a traditional manner via a relational database. The quantity of data that is presumed to imply Big Data is petabytes or more.

2. **Big Data is Evolving Faster As a Concept Than As a Working Infrastructure** The problems we've experienced in the past with storing, securing, sharing and making meaning of data are exacerbated in the current world of Big Data. Issues and struggles we experienced are magnified in the world of Big Data, accompanied by a growing set of data collection and storage technologies that are not yet up to the task of being able to properly protect the sensitive data contained therein. Older security models may not be enough or sufficient to properly care for the Big Data that is being collected.

3. **Hadoop is Big and Getting Bigger** New information technology that originates from the open source software world has been developed to work with Big Data. This new technology, called Hadoop, is capable of enabling the processing of very large quantities of data. Hadoop has created incredible opportunity to reveal more of the unknown in Big Data through the ability to bring so many more pieces of the puzzle together and serve it up ready for analytics engineers. One downside to Hadoop is that Hadoop databases typically have very slow processing rates, an artifact of current architectures. However, there are many powerful Hadoop-specific analytical tools that have been developed that are capable of processing and gleaning innovative meaning from these hoards of Big Data in a faster way.

4. **We May Need New Models for Database Security** Past models of secure schemes for entire data bases may be too costly for today's Big Data. However, the same theories, rules, and technologies apply to data today, even though the quantity of data has grown exponentially. We need to remind ourselves of the basics of data protection, both for security and privacy's sake, and that if they are performed well "in the small", they can be performed well "in the large" i.e. in the new world of Big Data. As is typical for technology that rises from within the open source community, the functional capabilities to work with Hadoop databases have developed far more quickly than the associated technology to control or protect the security and privacy of this data. If we did not perform these data protection tasks well in the past, didn't take proper care of our earlier data stores, how are we to secure and care for the Big Data we have in hand today, in a less mature, open source technology framework?

5. **Big Data Requires More Protection** Given the current state of technology and controls available today, it is easy for Big Data to quickly become a big problem with a really big price tag. Current issues we see today, where companies already do not sufficiently protect their data, lead to law suits, negative publicity, brand damage, and, possibly, regulatory fines and other fees. The more data that exists, the MORE data protection is required. Big Data requires newer security and privacy models that scale with Big Data, including both the ability to control access to data that is held within your networks, and providing protection to data that is leaving your networks.

6. **Surgical Application of Better Protection** In the current world of Big Data, secure practices and technologies may need to be applied in a more surgical manner to maintain the cost of implementing and maintaining the protection. There are costs to acquire the data, costs to maintain the data, costs to secure the data, as well as the cost to use and get value from the data. At the same time, there is an even stronger need to handle data and perform the basics of identifying, authenticating, authorizing and controlling access to data in the Big Data world. Applying strong data protection for all of your data can be very costly and cumbersome, with limited extra value or return on your investment. A need exists to implement stronger data protection for Big Data exactly where and when it is needed and to accept the costs of that stronger data protection.

7. **Know the Value of Your Data** It is vital that we truly understand the nature and sensitivity of the data in hand. Not all data are created equal. Some data is more liable to place an organization at risk, some data is more sensitive than others. Encryption of data can slow down performance, increasing latency and time-to-value on the data. This is even truer with the current state of Big Data technology. Hadoop technology is inherently slow, and imagine placing the additional burden of encryption into this same mix. Encryption can be applied to Big Data, and it's recommended to encrypt only the most sensitive data components within your Hadoop infrastructure—to not encrypt non-sensitive Big Data. As well, Big Data system back-ends, need to be protected in the manner of permitting only limited or no access to raw data by applications or services. For more real-time analysis, utilization, or reporting of data, it is recommended to use a relational database on the front-end of your Big Data back-end.

8. **Investment Drives the Need to Derive Meaning** Given the expense that businesses have likely invested to collect and store their tremendously large amounts of data, pressures to produce answers build within business organizations as they attempt to derive meaning from their data through analyzing their Big Data stores, looking for meaningful relationships via analytical tools to reveal correlations or repeatable patterns.

9. **New Old Career Opportunities** Gleaning meaning from Big Data means greater investment in decision-making algorithms and correlation engines. Data science, once an old career, is suddenly new again, and job candidates are sought with high priced compensation packages. It is a new model for "mentalists" who can see all by drawing meaning from mega data stores.

10. **Privacy Engineers are Vital** Remember, data is still data. You must know your data, the credibility of sources and frequency of update of your data. And, with Big Data, the value of data is growing at the same logarithmic rate as its size. It is important to focus on what truly needs to be protected, and at what level. To manage both cost and performance degradation, it is recommended that the

> Privacy Engineer spend energy on protecting the data within the Big Data store that is related to true risk or compliance. We need to mega-protect only what needs to be mega-protected. There are new technologies to facilitate the handling, correlation and making of meaning of Big Data. However these new technologies have grown and expanded far more quickly than the controls to maintain the protection of the data. It is vital to balance prudence, care, fiduciary obligation, and enablement. Just because we can, doesn't mean that we should.

Stage 4: Complete System Development

The development team will take the approved development prototype and complete the system development as soon as the prototype becomes accepted as a basis for the production system. The prototype caveats can provide elements of the completion criteria.

Stages 5 and 6: Quality Assurance and Rollout

Develop and Execute Test Cases

Test cases are developed for each use case, based on the activity diagram, the use case metadata, the sequence diagrams, and the class model. The supporting metadata test cases contain the following information:

- The application name
- The use case name
- The use case code (ID)
- Hardware or software
- The tester name(s)
- The date completed
- Test scenarios
- Within each test activity, test cases, and test conditions

The FIPPS/GAPP or similar principles will be used as test case criteria, along with other use case requirements. Chapter 10 contains a privacy question and answer checklist.

As construction builds are developed, the various components are integrated. The quality assurance project team members utilizes test cases, as described previously. As defects are found, they should be systematically documented.

Testing and Rollout Deliverables

Rollouts may be pilot operations or incremental implementations. Testing and rollout deliverables are:

- Test cases or scenarios based on data metadata business rules
- Test cases or scenarios based on use cases; activity, collaboration, and sequence diagrams; and supporting metadata
- Onsite and remote tests
- Defect list, including resolutions
- User acceptance tests
- Incremental rollout plans

Knowledge Transfer

Knowledge transfer to client personnel is critical for the effective transition of the application to the deployment and maintenance teams. It facilitates quick system problem resolution when issues arise in production and ensures system extensibility when additional functionality is needed.

Concerning the privacy component delivered alone, the training will be technical for most direct users. Business stakeholders and management will need to be made aware of the functionality and the potential impact of the privacy component.

For scenario 3 Vacation Planner Application, the training would focus on the functionality of the embedding system. The development team will need to be made aware of the functionality of the privacy component and its interface with an embedding system. The business team and management will need to understand the privacy rules enforced within the specific system.

For the scenarios, the use case requirements' specifications will provide the basis for the content of the subject matter within the training materials. This content may be presented as a white paper or an online training class or within a classroom setting. The students will be made aware of the privacy requirements being satisfied by the solution along with the business and technology aspects of the solution.

A key to implementing a successful privacy program is empowering employees and stakeholders of the organization to assist the company in preventing privacy problems. If privacy education and training are provided to the entire population within the organization, they will come to understand the fundamentals of privacy so they can help protect against privacy vulnerabilities. More advanced instruction can be provided to key people within the organization whose duties involve more exposure to systems or processes that implicate privacy information implementation.

Internal communications such as published guidelines, FAQs, and other documents are good ways to leverage the resources within the privacy team so that a broad audience can be reached efficiently. These published communications also provide a good starting point for new employees who need to quickly understand the important elements of the privacy policy. Ongoing internal training events can provide another

way to educate many at the same time. Problem-solving exercises involving practical scenarios can be very effective in getting active learning participation in internal training events.

Conclusion

This chapter has presented a systems engineering lifecycle methodology adapted to implement privacy engineering. This methodology has been used successfully for over 30 years, with the privacy adaptations being used in recent years. The use of models and modeling is crucial to intelligent systems design. The international standard UML was selected because it is a widely used standard and it covers object, data, and process modeling. Chapters 7, 8, and 9 will provide practical examples using this methodology and discussed in more detail.